Automated mail reply system

Sruti Kumari
112026600

Mohit Marwari
112027274

Debesh Mohanty
111973815

Abstract

In todays world time is the most important resource. For many this time might be insufficient to carry out their daily works in a professional or personal environment. In a professional environment most of the communication is through emails as professional environment demands written formal communication. To save time and increase throughput we try to develop an automated email response generator which will provide the user a set of possible responses as a reply to an email. The user just now has to choose a response instead of manually typing the whole response to the email. We were motivated from various papers and web pages among which the most informative seemed to be Google Inboxs implementation of automated email response. We hope our implementation will address the gaps of these implementations to some extent.

1 Introduction

In today's world, even after the advent of social networking, a majority of the communication still happens via electronic mails or e-mails. There is a continuous increase in the number of mails being sent everyday. Especially with the concept of enterprises being located in various geographic locations, emails have become an important medium of communication between the employees across various offices. A study showed that nowadays an average person receives around 97 business and around 88 personal mails spending 28% of his work time in reading and responding to emails. Especially for an employee of an organization, reading and replying to these mails take up 30 per cent of their daily time. Here in lies the importance of an automatic email reply system, which can address this daunting task and save the time of an employee and increase its productivity. This is an interesting problem to solve as not much previous work in NLP had directly aimed to generate automatic email reply suggestions. There were some previous works in this domain, but that mostly included baseline models based on Bayessian Network. In the neural world, the only notable work in this field was by Google in their Smart Reply paper which describes in some extent the model behind the New Smart reply feature in Google Inbox. But Google did not release any code base or documentation on how to build this system. It only provided a basic overview of their models and the challenges they addressed to ship this to production environment.

2 Task and Related Work

One of the biggest step in this direction is taken by Google as Smart Reply system. It is currently used in Inbox by Gmail and is responsible for assisting with 10 percent of all mobile responses. Another notable work in this direction is Awkwardly: A Response Suggester CS224N Final Project which uses Twitter Corpus to generate a pool of responses using various seq2seq LSTM neural network architectures and rank the responses using semantic intent clustering. Broadly, we can classify emails into two categories: informational and non-informational. Unlike other two papers mentioned above, that addresses generating responses for non-informational emails, we came across one more interesting work DejaVu: A Case for Assisted Email Replies on Smartphones which aims at complimenting them with informational content. In our project we have also tried to train our model to generate mail replies for informational mails also. We have described our model in detail in the Approach section below.
From the observations we made on the Googles

Smart Reply system, we observed that Google generate email responses for inputs mails like: Does tomorrow 5pm works fine with you for the meeting?; to which the Smart Reply system will generate a set of responses like: Yes, I will be there, No, I wont be able to make it, Maybe, will confirm later. The user may decide to use one of these three generated recommendations as a reply or may type in a response by himself. But the Smart Reply wont generate replies for input sentences like: What was the agenda of our last meeting?. On the other end, informational mail systems like the ones implemented on the DejaVu paper, it will generate a brief answer based on the users other mails history. All of the works mentioned above aims at either to generate the reply for either the informational or non-informational emails. We will start with generating responses for non-informational emails. But our training dataset is a database of over 500,000 emails generated by 158 employees of the Enron Corporation, so it is possible to extract some informational data related to business domain and predict responses based on that.

3 Our Approach

In this section, we will describe in details what approaches we took at every step of our project. We will start with cleaning our dataset and extracting useful information from it for our model. Next, we will show how we generated embeddings for emails. In the next section we will describe how we implemented our training and testing model followed by our experimental setup and analysis of result.

3.1 Dataset and Preprocessing

For training our model we are using the ENRON dataset. As we know email contains lot of fields like To, $From$, $Subject$, $Body$. First step in our project was to extract email body from the entire emails. Next we extracted the mail and their response pairs from the dataset. For which we kept emails with subject starting with "Re:" as reply to mail that has the characters following "Re:" as it subject. Once we had email response pairs, next step was cleaning up the data and we started with summarizing emails into 2 sentence and responses into 1 sentence.

Since we are dealing with emails which had large paragraphs or multiple paragraphs its impractical to use those mails for training our LSTM model. We needed to convert the emails to a summarized format which contains the relevant information in a shorter amount of sentences. Summarization of text is a vast research field but due to the scope and time for our project we could not explore more on this field so we sticked to a simpler way for summarizing emails. We take an input text mail and estimate frequencies for all the words in this mail and calculate a score for each word using the words frequency and and the maximum frequency that a word gets. We set score of all the stop words to zero. We now calculate scores of sentences which is the summation of scores of the words in that sentence. We then sort the sentences or store it in a priority queue and retrieve top two or three sentences. For emails we considered top two sentences as its summary and for responses we chose only one top score sentence as its summary. Selecting two sentences for summary of emails ensures that we have enough context as might not be if we had only one sentence. On the other hand, having only one sentence in response is done because ultimate aim of our project is to train the model to predict short one sentence replies, which can be done only if our training data has similar instances.

3.2 Generating Embeddings

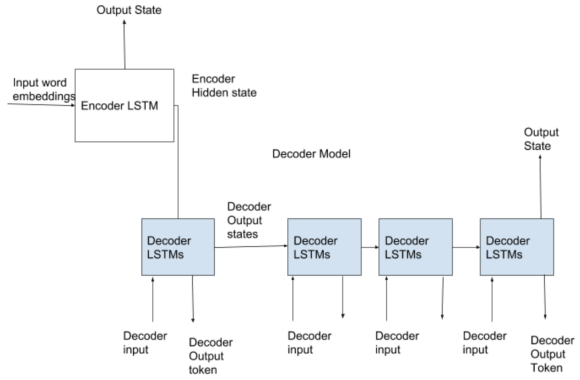
For each word in our mail we took its corresponding GloVe embedding and appended them together to generate one embedding for a mail/response. To generate individual word embeddings, we used StandfordNLP tool and ran that on our entire corpus.

3.3 Training

We split our data into training and test set for our model. For training we used a Sequence to Sequence LSTM encoder-decoder model. We used the methods provided by 'keras' to implement this. The encoder model takes embedding for each sentence token by token. The hidden state of last LSTM cell of encoder model is passed to first step of decoder model. Every cell of decoder model takes the hidden state of previous cell and the decoder input and generates a hidden state and output. Decoder input is the reply of that will email till that token.

One more step in our training phase was clustering. So, we cluster the embeddings of original

replies of the emails in our training data into different clusters. For this we used KMeans clustering from sklearn libraries. We save this model in our training phase and load the model during testing to evaluate how close our predictions are to the original replies.



3.4 Testing

For testing, first we summarize the emails and replies before we pass it to our model. In our case, we have use the same summarized data to split into test and train, so we don't need to do it again but if we had generate responses for an email not in our dataset, we first start by summarizing it and then pass it to model as described next. we again call the seq2seq model created in training phase. We pass the mail for which we have to generate the reply to encoder model and use its hidden state as initial state of decoder and the output generate by decoder till now is passed as input to the next decoder cell. Initially, we pass an empty string as input to first cell of decoder model. Once, we have generated the response, we need to figure out if the response generated is good enough. Given the resources we had we cannot afford manual intervention at large scale to validate the result, if they are meaningful or not. Hence, we used clustering model built in training module. We predict which cluster the response generated by our encoder-decoder model belongs to and also to which cluster the original reply belongs to and use this cluster prediction to this belongs to. We use these predictions to find the accuracy.

4 Code Section

Our code is divided into different modules. We have a module $makeData.py$ for generating email response pair into json file. We have a module named $summarize.py$ that generates summarized form of each mail. We have module $trainer.py$ that trains the model and saves it. We have a module $tester.py$ that test the model saved.

We have uploaded our code here on: github.com. More details on how to run the code is specified in the README file in our codebase.

5 Evaluation

5.1 DataSet

For our training and testing purposes we have used ENRON dataset. ENRON dataset consists of around .5M mails for around 150 employees. This dataset was made public during their fraud investigation case. And this perhaps, is the only dataset we found for emails that was this big to adequately train our model. This dataset contains 0.5Million messages and after our preprocessing, it had approximately 3700 email response pairs.

5.2 Experiments

We tested our models by changing the number of epochs, starting from 1, then 5,10,100,500,1000 and 2000. As expected, increasing the number of epochs generated more meaningful responses. Also, first we ran the model with 1000 email response pairs. Then, we generated email response pairs for entire data set and it was approximately 3700, then we train our model with 3700 email response pairs. We also tried trying different clustering algorithm for predictions like KMeans and Mean Shift Clustering and by varying number of clusters for each algorithm. Along with above experiments, we also tried tuning experiments with hyper parameters like changing embedding size of glove embeddings, dimension of hidden layer output in LSTM model, number of hidden layers in encoding model of LSTM,etc

6 Result and Analysis

As expected, our model is able to generate responses for mails in test set. Here are some of the samples generated by our model:
Email: This is a very important file so I need help as soon as possible I somehow created two of the same files that are linked and I cannot unlinkclosing excel but when I re open xxx4 both the files are re opened.
Response: call recipient

Email: The utilities are already in her name In my mind the utilities were never a bargaining chip If it is acceptable please sign and fax to her directly at830 372 5400
Response: please contr

Next, we present what accuracy we received in cluster prediction for different number of epochs and different number of email response pairs.

Epochs #	No. of email reply pairs	Accuracy
1000	1000	14%
1000	3700	28%
3000	1000	42%
3000	3700	53%

7 Challenges

The ENRON dataset came up with its own problems. First of all, the structure of the ENRON dataset was a problem for us. Each folder inside this dataset corresponded to an employee and each sub folder inside this contained the mail category structure he had (inbox, outbox, trash, spam, meetings etc). So before the preprocessing part of this dataset, we had to recursively go through each folder, extract the mail and save it to a text file. This text file contained all the email texts for the entire dataset. This amounted to around 2.2GBs. This was a bit time consuming. Also the message texts were in raw format i.e. it had all the header information along with the main content that we were interest in. Even after removing the header information, the size of the file was still around 1.6GBs. To generate each set of mail and its reply pair mail; the conversation threads also took an enormous amount of time. Even after generating the mail reply pairs, we found out that the mail corpus had further issues, some of the sentences had characters that were not encoded in UTF-8 format. And many of the mail reply pairs were 4-5 sentences longer. For the purpose of our model training, we did not want our mail corpus to have such long sentences.
Given the capacity of the machines that we had the training took a lot of time. High end systems with GPU could help in training the model faster and analyzing the result. As a workaround, we tried to run our code on Google Colab.
One of the major challenge in implementing this project was that we could not find much work in this field. Although, Google has implemented it in its Inbox but not much documentation is there for someone to pick it up and build on top of that. There were many works related to chat bot, so we referred to those works and the Google smart reply and tried to implement a basic model to address this kind of problem.

8 Conclusion and Future Works

So we were eventually able to construct a encoder decoder LSTM model that predicted responses to an incoming mail. Some of these responses were meaningful; while some were not. One of the biggest reason was that the dataset was noisy; but however, since it was a mail corpus that is encountered in daily life, so we can not expect the users to write all mails that are grammatically correct. We need some mechanism to give correct responses even there is noise in the training data. Some of the improvements that we suggest are summarized in the below paragraph.

To summarize long paragraph mails we are using the word frequencies as a determiner to extract the most relevant 2 sentences in the paragraph (not including the stop words). It is a simple technique but we this frequency based approach may not work in all cases. So we could improve on this part so as to get meaningful mails even after summarizing. Often summarizing also leads to loosing some important data, so we came across a technique called data augmentation. Data augmentation adds value to base data by adding information derived from internal and external sources within an enterprise. So we could use summarizing in conjunction with data augmentation so as to ensure we get smaller emails without loosing valuable information.
Another point where we think we could improve is the embeddings that we send to the model. Currently, we send only word embeddings to the LSTM encoder-decoder model, we could add more information like POS tags, dependency relations from dependency parse trees and send that to decoder. This would help generate more meaningful replies, as it is not being done by our model currently in all cases.

References

[1] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann (2017) *Smart Reply: Automated*

Response Suggestion for Email, Google.

[2]Abdulkareem Al-Alwani *A Novel Email Response Algorithm For Email Management Systems*, Journal of Computer Science 10

[3] Uma Parthavi Moravapalle, Raghupathy Sivakumar *DejaVu: A Case for Assisted Email Replies on Smartphones*, 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)

[4] Quinlan Jung, Kai-Cheh Huang *Awkwardly: A Response Suggester CS224N Final Project*